

INTRODUCTION TO PROGRAMMING
AN IBM 360/25
IN "BASIC FORTRAN IV" LANGUAGE

F. HEGYI

FOREST RESEARCH LABORATORY
ONTARIO REGION
SAULT STE. MARIE, ONTARIO

INFORMATION REPORT 0-X-152

CANADIAN FORESTRY SERVICE
DEPARTMENT OF FISHERIES AND FORESTRY
JUNE 1971

*Copies of this report may be obtained
from*

*Director, Ontario Region,
Canada Department of Fisheries and
Forestry,
Box 490, Sault Ste. Marie, Ontario*



Frontispiece. An IBM 360/25 system, showing console typewriter, line printer and card read/punch unit.

ABSTRACT

This report introduces programming an IBM 360/25 computer in "Basic FORTRAN IV" language. Computer hardware is briefly explained, and the fundamentals of the FORTRAN language are dealt with in detail. Topics covered in the text include: arithmetic expressions, arithmetic statements, input/output statements, control statements (GO TO, IF, Computed GO TO, and DO), and specification statements. A brief introduction is also given to the Disk Operating System and to the set-up of the control cards needed for various types of jobs.

The text was compiled for a seminar course given by the author for the staff of the Ontario Region, Canadian Forestry Service, during the Fall of 1970.

TABLE OF CONTENTS

	<i>Page</i>
INTRODUCTION	1
BASIC CONCEPTS	1
<i>Magnetic core storage</i>	1
<i>Binary numbers and bytes</i>	1
<i>Registers and control</i>	2
<i>System 360 model 25</i>	3
FUNDAMENTALS OF THE "BASIC FORTRAN IV" LANGUAGE	4
<i>Constants</i>	5
<i>Variables</i>	7
<i>Arithmetic expressions</i>	8
<i>Rules for constructing arithmetic expressions</i>	9
<i>Arithmetic assignment statement</i>	10
INPUT AND OUTPUT	11
<i>Example of a small FORTRAN program</i>	15
CONTROL STATEMENTS	15
<i>Unconditional "GO TO" statement</i>	15
<i>Arithmetic "IF" statement</i>	16
<i>"CONTINUE" statement</i>	16
<i>"END" statement</i>	17
<i>"STOP" statement</i>	17
<i>"PAUSE" statement</i>	17
<i>Computed "GO TO" statement</i>	18
<i>"DO" statement</i>	18
<i>Rules for DO looping</i>	20
SPECIFICATION STATEMENTS	21
<i>DIMENSION statement</i>	22
<i>INTEGER specification statement</i>	24
<i>REAL specification statement</i>	25
<i>DOUBLE PRECISION specification statement</i>	25
SOME FORTRAN LIBRARY SUBPROGRAMS	25

TABLE OF CONTENTS

	<i>Page</i>
PROGRAMMING CONSIDERATIONS	27
DISK OPERATING SYSTEM	27
DEFINE FILE <i>statement</i>	28
<i>Direct Access</i> WRITE <i>statement</i>	30
<i>Direct Access</i> READ <i>statement</i>	30
BIBLIOGRAPHY	35
APPENDIX A	39
APPENDIX B	42
APPENDIX C	45
APPENDIX D	47

INTRODUCTION

In scientific data processing, one is often confronted with highly complex problems whose solutions could take several man-years even with a calculator. In such cases, one would most likely turn to an electronic assistant, which is called "the electronic computer". An electronic computer works in a somewhat similar manner to a calculator. An ordinary calculator has one or two registers in which either data are stored or arithmetic operations such as addition, subtraction, multiplication, or division are carried out. In a calculator with two registers, the contents of one register may be added to the contents of another register to obtain the sum of the two numbers. In a computer, there are several registers as well as thousands of storage locations where numbers or instructions may be stored. Each storage location is identified by a numeric address. A number or a variable stored at a specified storage location may be brought into one of the registers, and then another number--stored at a different location--may be added to it. This is, in effect, just a more sophisticated form of the operation carried out with an ordinary calculating machine.

BASIC CONCEPTS

Magnetic core storage

Computers function in a binary mode, that is, they operate on quantities that can have only two possible states--the "on" and "off" or 1 and 0 states. A device that can take only one of two possible states is referred to as a binary indicator. In most IBM 360 models, thousands of ferric oxide rings (19 mils inside diameter) are used as binary indicators. These rings are easily magnetized, can retain their magnetism indefinitely, and can be just as easily demagnetized. Thus, they are called magnetic cores or often are referred to as the memory of the computer. Magnetic cores are arranged in planes and are strung like beads on wires. They may be magnetized by passing an electric current through the wire. The direction of the magnetic field depends on the direction of the current. It is agreed by convention that cores magnetized in one direction represent 1's, and those magnetized in the opposite direction represent 0's.

Binary numbers and bytes

To represent decimal numbers in binary mode, a combination of 1's and 0's is used, i.e., cores magnetized and not magnetized. A binary digit is also called a bit. The smallest unit of information with which the computer deals is a byte, which is composed of nine bits--eight bits to represent information and an extra bit called a parity bit. This

extra bit is used for the detection of possible machine malfunctions. With the eight bits, $2^8 = 256$ distinct characters can be represented while using only one position of storage. To represent a constant or a variable in an IBM 360/25 computer, four storage locations or four bytes are needed. Thus, $4 \times 8 = 32$ binary digits (not counting parity bit) are taken up by each constant or variable. Because of this, writing in binary notation may be tedious and subject to clerical errors. Various notations have been developed to represent binary numbers in a more compact notation, of which the hexadecimal one has become standard for the IBM 360/25. Hexadecimal notation is based upon powers of 16 and uses 16 different digits: 0, 1, 2, 3, 4, 5, 6, 7, 8, 9, A, B, C, D, E, F, (where, A = 10, B = 11, ..., F = 15).

For example,

Decimal	Hexa- Decimal	Binary	Decimal	Hexa- Decimal	Binary
0	00	00000000	13	0D	00001101
1	01	00000001	14	0E	00001110
2	02	00000010	15	0F	00001111
3	03	00000011	16	10	00010000
4	04	00000100	17	11	00010001
5	05	00000101	18	12	00010010
6	06	00000110	.	.	.
7	07	00000111	.	.	.
8	08	00001000	100	64	01100100
9	09	00001001	.	.	.
10	0A	00001010	200	C8	11001000
11	0B	00001011	.	.	.
12	0C	00001100	255	FF	11111111

Registers and control

Besides magnetic core storage, the IBM 360/25 computer also makes use of registers. Registers are electronic units which hold information temporarily during the processing of a program. There are three types of registers in the 360/25: the general purpose (used for storing the integer operand involved in arithmetic operations), the floating point (used for storing the operands of floating point numbers), and the control registers (used as "electronic scratch pads").

The computer operates under the control of instructions. The instructions might be in the form:

move into register 7 the number at location 0140, then add to register 7 the number at location 0132.

This, in hexadecimal form, would look as follows:

58 7 0140
5A 7 0132

where: 58 and 5A are instruction codes (meaning load and add, respectively), 7 is the register number, and 0140 and 0132 are core address numbers. Thus, the computer receives the instructions in coded numeric form.

System 360 model 25

The IBM system 360/25 may simply be thought of in terms of three units:

- (1) the storage unit, containing the main core storage,
- (2) the central processing unit (C.P.U.), containing the circuitry for performing arithmetic operations and control (also the various registers),
- (3) the input/output units, providing means of communication with the computer. Although there may be several input/output units attached to an IBM 360 system, generally only four are attached to a model 25 which supports basic FORTRAN IV (with disk operating system). These are:
 - (a) A console typewriter, which is a modified IBM selectric typewriter, used mainly for brief communications between the operator and the computer, such as starting or stopping program runs. It also provides a log of what the computer has done.
 - (b) A card read/punch unit, which provides for reading information from cards into core storage and for punching the contents of specified core areas on cards. In scientific data processing, where often a large number of cards must be read, it is an economic advantage to have a relatively fast card reader attached to the computer, such as the model 2540 (which reads 1000 cards/minute).
 - (c) A line printer, which prints the contents of specified core areas on paper. Generally, an entire line, consisting of 120-144 characters, is printed at once. Again, in scientific data processing, speed is important. Hence, the model 1403 line printer, which can print between 600 and 1000 lines per minute, is often used with a model 25 computer.

(d) A disk drive (or in some cases two or more disk drives), which in effect expands the capacity of a computer at a relatively low cost. This disk unit looks something like a juke box. The disk is covered with magnetic coating and information from core storage is recorded on it in much the same way as a phonograph record is made. Later on, this information may be read back into core storage and used either in calculations or directly for printing.

FUNDAMENTALS OF THE "BASIC FORTRAN IV" LANGUAGE

The word "FORTRAN" is derived from "FOR"mula "TRAN"slation, and this is what it actually means. FORTRAN language is a cross between English and mathematics; therefore, it is mainly used for solving scientific problems. In a way, it is independent of the type of machine, or in other words, is applicable to different makes of computers. However, each machine has a certain dialect, or more correctly, a number of restrictions, but once the basic elements of the FORTRAN language are mastered, it is relatively simple to program the different makes of computers.

A program written in Basic FORTRAN IV language consists of a set of statements written for the purpose of directing the computer through the various steps of the program. Basically, the computer may be instructed through binary or hexadecimal numbers. That would, however, be very confusing and time consuming. Thus, the FORTRAN language provides a practical approach and an easy access to the computer.

A program written in FORTRAN is called the source program. It must be translated into machine language in order to be suitable for execution on an IBM 360 system. This translation is performed by another program, called the FORTRAN compiler, which is generally built into the system and the reader need not be concerned with it. For the sake of interest, it may be mentioned that when a source program is translated into machine language it is called an object program.

In order to be able to read and write in any language, it is best to learn its alphabet first, then something about its grammar, and finally to build up a fairly good vocabulary. This logical course of action also follows in learning to read and write in FORTRAN language. Here, the alphabet is the same as in English, (A to Z) plus the \$ sign, but the alphabetic characters must be capital letters. The numeric characters run from 0 to 9, and in addition, special characters such as + - / = . () * , ' & and blank are acceptable in FORTRAN. Thus, in FORTRAN, there are alphabetic, numeric, and special characters. When alphabetic and numeric characters are mixed within a word, the word is called alphameric.

The basic language units used for constructing FORTRAN statements are constants, variables, operations, expressions, and functions.

Constants

Any number that appears in a source statement in digit form is called a constant. It is a fixed, unvarying quantity. Three types of constants can be used in basic FORTRAN: integer, real, and double precision.

(1) An integer constant is a whole number written without a decimal point. It may be zero or any positive or negative number of less than approximately 10 decimal digits (maximum magnitude = 2147483647 or $2^{31} - 1$), and it must not contain embedded commas. If a constant is unsigned, its positive value is assumed. An integer constant occupies four locations of storage, that is, four bytes.

Examples:

Valid integer constants	Invalid integer constants
0	0. (decimal)
+1	5. (decimal)
1	1,555 (comma)
-1	2147483648 (too large)
9785	0.25 (decimal)
-9785	-0.25 (decimal)
78	.0005 (decimal)
1777777	-10,100 (comma)
-1777777	22147483748 (too large)

(2) A real constant is any number written with a decimal point. It may be zero or any positive or negative number with a maximum of seven significant digits. A real constant may explicitly be specified as real by appending an exponent to it. The exponent consists of the letter E followed by a signed or unsigned 1- or 2-digit integer constant. The letter E indicates that the real number is to be multiplied by the specified power of 10 (the integer constant following E specifies the power of 10). A real constant also occupies four locations of storage, that is, four bytes. It may not contain embedded commas. An integer constant followed by a decimal exponent is also considered real. The range of exponents of a real constant is from 10^{-78} to 10^{75} .

Examples:**Valid real constants**

```
+0.
-111.11
2.12345
0.0
+1.0
1E3           i.e. 1.x103=1000.
3.4578
.000256
5.2E+0       i.e. 5.2x100=5.2
5.2E00        i.e. 5.2x100=5.2
5.2E03        i.e. 5.2x103=5200.
-5.2E03       i.e. -5.2x103=-5200.
5.2E-2        i.e. 5.2x10-2= 0.052
```

Invalid real constants

```
5,271.        (embedded comma)
2.E           (missing the integer constant after E)
2.E222        (E is followed by a 3-digit integer)
2.E-80         (magnitude is outside the allowable range)
1.E2.          (the constant after E is not integer)
0              (missing a decimal point)
```

(3) A double-precision constant differs from a real constant mainly in terms of precision. It can contain from 8 to 16 significant digits, and the exponent following it is D instead of E.

Examples:**Valid double-precision constants**

```
2.12345678901234
2.0000000
0.0D0
2.5D07         i.e. 25000000.
-0.222D-3      i.e. -.000222
7.8D0          i.e. 7.8 x 100 = 7.8
```

Invalid double-precision constants

```
222,511.2      (embedded comma)
1234567        (decimal point missing)
22.2D-95        (magnitude outside the allowable range)
5.2D4.          (not an integer exponent)
```

Variables

A variable in FORTRAN language is a symbolic representation of a quantity; or alternatively, a number referred to by name rather than by value is a variable. Names assigned to variables must consist of from one to six alphabetic characters, i.e., alphabetic--A through Z, including \$ sign--and numeric--0 through 9, and the first one must be alphabetic. Special characters are not permitted in FORTRAN names.

The type of a variable corresponds to the type of the data it represents--it may be in integer, real, or double-precision mode. The mode of a variable is specified by the first character of its symbolic name. Thus, if the first letter of a variable's name is either I, J, K, L, M, or N, it is in integer mode; that is, it represents an integer quantity. If the symbolic name of a variable does not start with any of the above six letters, then it may either be in real or double-precision mode. It is double precision only when it is explicitly specified with a 'D' exponent, or when it is specified at the beginning of the program.

Examples:

Valid integer names	Valid real names
IBM360	TREE
IRENE	AIBM
IVAN	DIAM
IDIAM	A
ITREE	\$GOTU
KAREA	SILLY
L100	OHBOY
M2F4\$	WHATEH
N	FUNNY
NOMORE	TRY

Invalid integer and real names

K735\$75	(more than six characters)
WRONG.	(contains a special character)
5IBM	(first character is not alphabetic)

In the FORTRAN language, a set of variables identified by a single variable name is called an array. For example, let the waist measurement of three young ladies be called WAIST1, WAIST2, and WAIST3, where WAIST1 refers to the first, WAIST2 to the second, and WAIST3 to the third young lady. Thus, the variable names imply that the three waist measurements are in "real" mode. If it is further specified that the three waists in question have been measured and found to be 23, 32, and 24 inches, respectively, in FORTRAN arithmetic, they may be written as

```
WAIST1 = 23.0
WAIST2 = 32.0
WAIST3 = 24.0
```

However, these three waist measurements may be identified by a single name, say WAIST. To be able to refer to each individual measurement separately, an integer quantity (in parentheses) is attached after the name. Thus,

```
WAIST(1) = 23.0
WAIST(2) = 32.0
WAIST(3) = 24.0
```

The integer quantity in parentheses is called the subscript of the variable. A variable name with a subscript immediately following it is called a subscripted variable. The subscript may be an integer constant or an integer variable. For example, WAIST(I) refers to the I-th value of the variable WAIST, and this value could be 1, 2, or 3. For example,

```
if I = 1
    WAIST(I) = 23.0      that is, WAIST(1) = 23.0
or if I = 3
    WAIST(I) = 24.0
```

The foregoing is only a brief introduction to subscripted variables, which will be dealt with later on in more detail.

Now, try to solve Exercises 1 to 4 in Appendix A.

Arithmetic expressions

Basic FORTRAN IV provides only one type of expression, the arithmetic expression. An arithmetic expression is a sequence of constants, variables, and functions separated by operation symbols and parentheses to form a meaningful mathematical expression. In addition, a single constant or a variable can also be an arithmetic expression. The following five operations are permitted:

- + denoting addition
- denoting subtraction
- * denoting multiplication
- / denoting division
- ** denoting exponentiation

Examples:

A+B	FRANK+TALK
3	2**2
2+1	Y/4.5
(A+B*C)	A-C
	1.D-02

Rules for constructing arithmetic expressions:

- (1) All desired operations must be specified by operation signs. For example, AB does not mean the multiplication of A and B. If multiplication is desired, it must be written as A*B or B*A.
- (2) When two operation symbols follow in succession, they must be separated by parentheses. For example, C*-2. is invalid, and should be written as C*(-2.).
- (3) Computation is performed from left to right according to the following hierarchy of operations:
 - (a) evaluation of functions (e.g. SIN(X))
 - (b) then **
 - (c) then * and /
 - (d) and finally + and -

This hierarchy is used to determine which of two consecutive operations is performed first until the end of the expression is encountered, and then all of the remaining operations are performed in reverse order. However, in the case of exponentiation, the evaluation is from right to left.

For example, the expression

$C*D/A-F^{**}E^{**}G+2.$

would be evaluated as

Step 1 $E^{**}G$
 Step 2 $F^{**}(\text{result of Step 1})$
 Step 3 $C*D$
 Step 4 $(\text{result of Step 3})/A$
 Step 5 $(\text{result of Step 4})-(\text{result of Step 2})$
 Step 6 $(\text{result of Step 5})+2.$

As in algebra, parentheses may be used in arithmetic expressions to specify the order of operations. For example,

$((C*D)/(A))-(F^{**}(E^{**}G)))+2.$

In fact when one is in doubt about the order of operations, the best policy is to make use of parentheses, even if it means inserting unneeded ones. When one or more pairs of parentheses are used, the expression within the parentheses, or within the innermost parentheses, is evaluated first.

- (4) The mode of the result of an operation depends on the mode of the operands involved in the operation. Mode is given for the internal representation of the result. The actual value is dependent upon the mode of the target variable. For example, the results of the

operations +, -, *, /, and ** would be as follows:

Mode of one operand	Mode of the other operand	Mode of the result
integer	integer	integer
real	real	real
double precision	double precision	double precision
integer	real	real
integer	double precision	double precision
real	double precision	double precision

Furthermore, it should be noted that a negative real or double precision quantity cannot be exponentiated, since the log of a negative number is complex.

Arithmetic assignment statement

The arithmetic assignment statement resembles the conventional algebraic equation. Its general form is

A=B

where: "A" is a variable name (without a sign), and
"B" is any FORTRAN expression.

In effect, the FORTRAN arithmetic assignment statement tells the computer to evaluate the arithmetic expression on the right of the equal sign, and then to assign the resulting value to the variable named on the left of the equal sign. Thus, the equal sign is not used here as it is in ordinary mathematical notation, since the left side of the statement must be the name of a single variable. The arithmetic assignment statement is sometimes abbreviated to arithmetic statement, and from now on in this text, it will be used in the latter form.

Example:

A=2.	means assign 2. to "A"
B=7.+8.	means add 8. to 7. then assign the result (15.) to "B"
I=3	means assign 3 to "I"
J=5	means assign 5 to "J"

K=I+J means add the value of "J" (=5) to the value of "I" (=3), then assign the result (=8) to "K".

In the previous section it was stated that the mode of an arithmetic expression depends on the mode of the operands. Generally, it is a good practice not to mix modes in arithmetic expressions. However, when modes are mixed, the computer changes the operands into a uniform mode; that is, if both integer and real quantities are present in an arithmetic expression, integer quantities are changed into real, then the indicated operations are performed. This also extends to arithmetic statements, where the mode of the result of the arithmetic expression is equated with the mode of the variable on the left of the equal sign.

Example:

I=5/2	here, 5/2 results 2.5 but in integer arithmetic, the decimal portion is "cut off" to give I=2
A=5*2	first, 5*2 is 10 (integer), then this integer value is converted into real, so A=10.0
J=2.7	the value 2.7 is truncated to an integer value, and this value will replace the value of J, so J=2 is the final result.

INPUT AND OUTPUT

The transfer and control of the flow of data between an input/output device and internal storage is performed in FORTRAN through the use of input/output statements. In an IBM 360/25 system, where data are frequently submitted into internal storage from punched cards as read by the card reader, the "READ" statement is the relevant input statement.

First, information needs to be put on the cards. This operation is performed by any card-punching machine, such as the IBM 029. A card generally has 80 columns that may be punched, and each column has 12 punching positions, one each for digits 0 to 9 and two zones above the zero. Numbers are recorded on cards by punching a single hole in the corresponding digit positions of the desired column. Alphabetic and special characters are the combinations of a digit and one or both of the zone-position punches. Hence, each numeric, alphabetic, and special character requires one column on the card. Figures 1 and 2 in Appendix B show examples of punched cards.

When punching FORTRAN source programs, the first five columns of the card are used exclusively for numbering the FORTRAN statements. The statements themselves are punched in columns 7 to 72, but if a statement

is too long for one card, it may be continued on as many as 19 successive cards by punching any character, other than blank or zero, in column 6 of each continuation card. Columns 73 to 80 are not significant to the FORTRAN compiler, and thus may be used for identification information. Furthermore, if "C" is punched in column 1, any information written in the rest of the columns will not be executed by the computer, but will be printed when the program is listed. A card such as this is called the comment card.

In contrast, when data are punched on cards, all 80 columns may be used.

Generally, FORTRAN source program cards, specifying certain operations which the programmer wishes to perform are read into the computer under the supervision of the operating system. On the other hand, instructions for the reading of data cards are given by the programmer in the source program. This may be done through a READ statement which has the following general form:

`READ(i,j)list`

where: "i" is an unsigned integer constant or an integer variable that represents a data-set reference number. The programmer can obtain the relevant number generally from the personnel of the computer centre. For example, for the IBM 360/25 used by the Ontario Region, the value of i is 1.

"j" is the statement number of the FORMAT statement used for describing the data being read.

"list" refers to a possible list of variable names, separated by commas.

The counter-part of the READ statement is the WRITE statement. This has the same general form as the READ statement, except that the word WRITE is substituted for the word READ. Therefore, its general form is:

`WRITE(i,j)list`

where: "i", "j", and "list" refer to data-set reference number, FORMAT statement number, and an optional list of variable names, respectively. Again, the value of "i" depends on the installation. For the IBM 360/25 used by the Ontario Region, the value of i is 3.

As indicated by the input statements, there is another statement, the FORMAT statement, which generally follows the input statements, i.e., not necessarily right after them, but it is included in the same program. The FORMAT statement has the following general form:

`j FORMAT(c1,c2,...,cn)`

where: "j" is the same number as specified in the READ and WRITE statements, and

"c₁,c₂,...,c_n" are format codes.

Format codes describe the type of data being transmitted. The following format codes are most often used.

(1) I format for integer data. Its general form is

In where "n" is an unsigned integer constant, less than or equal to 255, and specifies the number of characters of data. For example, to read 3520 from columns 1 to 4, the format code should be I4

(2) F format for real data without exponent. Its general form is

Fn.m where "n" is the same as above, and "m" is an unsigned integer constant specifying the number of decimal places to the right of the decimal point. For example, 2840.5 could be punched with the decimal point, or as 28405 in columns 1 to 5, i.e., without punching the decimal point, and the decimal point may be properly positioned through the format code as F5.1

(3) E format for real data with exponent. Its general form is

En.m where "n" and "m" mean the same thing as above.

(4) D format for double-precision data. Its general form is

Dn.m where "n" and "m" mean the same thing as above.

(5) A format for character fields. Its general form is

An where "n" again means the same as above. For example, to read the word LOVE from columns 1 to 4, the format code A4 may be used. An A format also transmits, besides alphabetic characters, numeric and special characters.

(6) H format for literal data. Its general form is

nH where "n" again has the same meaning as above, except that it refers to the number of characters in a text or in literal data. For example, the word LOVE transmitted in H format is 4HLOVE where 4 is referring to

the four characters (LOVE) following "H". Literal data may also be transmitted between a pair of apostrophes. For example, 'LOVE'. In both cases, blanks are classified as characters. The H format is also used for carriage control (page or line skipping). For example, when a line printer is used for output, the first byte of the output record is not printed, but used for controlling the action of the printer's carriage control tape. Usually, the H format is used for this purpose by writing 1H, where the character after "H" may be blank, 0, +, or 1, depending on the desired instruction. The meanings of the characters are as follows:

blank means single space before printing,
 0 means double space before printing,
 + means do not space before printing,
 1 means skip to the first line of the next page.
 Thus, 1H1, would instruct the computer to start printing on the first line of the next page. Furthermore, a / may also be used for skipping a card or line.

(7) X format for skipping columns and fields. It has the form

nX where "n" again means the same thing as in (1). For example, to skip five columns, 5X is the relevant format code.

(8) T format for transferring data. Its general form is

Tn where "n" is an unsigned integer constant designating a character position in a record where transfer of data is to start. For example, if T5,4HLOVE is designated in a FORMAT statement used by a WRITE statement, the printing of LOVE would start in position 4. It should be noted that in this case the first character is used for carriage control.

(9) n(..) indicates a group format specification. Here, "n" signifies the number of times the format codes inside the parentheses are repeated. For example, to read 21 35.0 12 and 71.2 from a card punched as 2135012712 in columns 1 to 10, the relevant format codes are 2(I2,F3.1). When only one format code is to be repeated, "n" is written in front of the particular format code. For example, 2I4 or 2F5.1.

Example of a small FORTRAN program

Calculate the area of a triangle.

First, read values for BASE and HEIGHT from a data card punched as 12052 in columns 1 to 5; the value for BASE is in columns 1 to 3 with one decimal, and the value for HEIGHT is in columns 4 to 5 with one decimal. Then, calculate the AREA by multiplying BASE with HEIGHT and dividing the product by C (where C is a real constant with a value 2.). Call the result "AREA".

```
C=2.
READ(1,1)BASE,HEIGHT
1 FORMAT(F3.1,F2.1)
  AREA=BASE*HEIGHT/C
  WRITE(3,2)AREA
2 FORMAT(1H1,26HAREA OF THE TRIANGLE IS = ,F8.2)
  END
```

This program is written on a FORTRAN coding sheet in Appendix B, Figure 3. Now, try to solve Exercises 5 to 7 in Appendix A.

CONTROL STATEMENTS

Generally, FORTRAN statements are executed sequentially; that is, control begins with the first statement, then passes from one statement to the next, unless a control statement alters that order. Alternatively, the order of the execution of FORTRAN statements may be controlled through control statements. These are as follows.

Unconditional "GO TO" statement

Its general form is

GO TO nnnnn

where: "nnnnn" is the number of an executable statement, to which control is to be passed. For example,

```
C=2.
3 READ(1,1)BASE,HEIGHT
1 FORMAT(F3.1,F2.1)
  AREA=BASE*HEIGHT/C
  WRITE(3,2)AREA
2 FORMAT(1H1,6HAREA= ,F8.2)
  GO TO 3
  END
```

This program causes the reading of the values for BASE and HEIGHT from a card, and then the calculation and the printing of AREA. After the execution of the GO TO statement, control is passed back to the READ statement, i.e., statement number 3. However, the program hangs up and never comes to a normal exit.

Arithmetic "IF" statement

Its general form is

IF(a)n₁,n₂,n₃

where: "a" is an arithmetic expression, and "n₁", "n₂", and "n₃" are the numbers of executable statements.

The "IF" statement causes control to be transferred to statement number "n₁", "n₂", or "n₃", when the value for "a" is less than, equal to, or greater than zero, respectively. Since control is being transferred from this control statement to a specified statement, the statement immediately following an IF statement must be numbered. For example,

```
C=2.
3 READ(1,1)BASE,HEIGHT
1 FORMAT(F3.1,F2.1)
IF(BASE)3,3,4
4 AREA=BASE*HEIGHT/C
WRITE(3,2)AREA
2 FORMAT(1H1,6HAREA= ,F8.2)
END
```

This program causes the reading of the values for BASE and HEIGHT from a card, and then tests the value of BASE. If a blank card is read (or BASE = 0.), the "IF" statement would cause the control to be transferred back to the READ statement. Thus, another card is read. If, on this card, the value for BASE is a positive number, the area of the triangle would be calculated and printed. Then, the program comes to a normal end. Note that if on the first card the value for BASE was a positive number, another card would not be read, and calculation would continue with the information obtained from this card.

"CONTINUE" statement

This is a dummy statement that may be placed anywhere in the program. The "CONTINUE" statement does not affect the sequence of execution in a source program, and is frequently used with other control statements. For example, in the AREA OF TRIANGLE program, it may be used as follows:

```

3 READ(1,1)BASE,HEIGHT
1 FORMAT(F3.1,F2.1)
  IF(BASE-99.9)4,5,4
4 AREA=BASE*HEIGHT/2.
  WRITE(3,2)AREA
2 FORMAT(1H1,'AREA= ',F8.2)
  GO TO 3
5 CONTINUE
END

```

This program causes the reading of a card containing the values for BASE and HEIGHT. Then, the "IF" statement tests whether BASE is equal to 99.9 or not. If BASE is not equal to 99.9, the area of the triangle is calculated and printed, after which, the "GO TO" statement transfers control back to the READ statement. This whole process is repeated until the computer reads a card with 999 in columns 1 to 3 (i.e., the columns allocated to the variable named BASE). At this step, the value of the arithmetic expression in the "IF" statement is $99.9 - 99.9 = 0$. Control is then transferred to statement number 5, and to the "END" statement. The card with 999 punched in columns 1 to 3 is called the stop card, since it causes the termination of card reading. Note that it is assumed that 99.9 would never be an observed value for BASE.

"END" statement

Its general form is simply END and it is a nonexecutable statement that defines the end of a source program or source subprogram for the compiler. The "END" statement must be the last statement of each program or subprogram.

"STOP" statement

Its general form is simply STOP or STOP n where "n" is an integer constant of not more than 5 digits. The "STOP" statement terminates the execution of the object program, and if "n" is specified, its value is typed on the output device(s).

"PAUSE" statement

Its general form is PAUSE or PAUSE n where "n" is an integer constant of not more than five digits. When a "PAUSE" statement is encountered, the program waits until operator intervention causes it to resume execution, starting with the next statement after it.

Computed "GO TO" statement

Its general form is

GO TO (n₁,n₂,n₃,...,n_n),i

where: "n₁","n₂","n₃","n_n" are the numbers of executable statements and "i" is a non-subscripted integer variable.

The computed "GO TO" statement causes control to be transferred to the statement numbered "n₁","n₂","n₃", or "n_n", depending on whether the current value of "i" is 1, 2, 3, ..., or "n", respectively. For example, a computed "GO TO" statement may be used in the AREA OF TRIANGLE program as follows:

```

9 ITEM=0
3 READ(1,1)BASE,HEIGHT
1 FORMAT(F3.1,F2.1)
  IF(BASE-99.9)4,5,4
4 AREA=BASE*HEIGHT/2.
  ITEM=ITEM+1
  GO TO (6,7,7,7),ITEM
6 WRITE(3,2)AREA
2 FORMAT(1H1,'AREA= ',F8.2)
  GO TO 3
7 WRITE(3,8)AREA
8 FORMAT(1H0,'AREA= ',F8.2)
  IF(ITEM-4)3,9,3
5 CONTINUE
END

```

The above program prints only four areas per page; that is, the computed "GO TO" statement causes the first AREA to be printed on a new page (because ITEM = 1 and so control is transferred to the first statement number, i.e., to statement number 6). Since ITEM is incremented to 2 before the computed "GO TO" statement is encountered again, control is transferred to statement number 7. Now, try to follow the various steps in the program, assuming it has 10 data cards. Also, try to solve Exercise 8 in Appendix A.

"DO" statement

Its general form is

DO n i=m₁,m₂,m₃

where: "n" is a statement number, "i" is a nonsubscripted integer variable, and "m₁","m₂","m₃" are unsigned integer constants

or nonsubscripted integer variables whose values are greater than zero.

The DO statement is a command to execute repeatedly the statements that physically follow the DO statement (up to and including the statement numbered "n"). These statements that are to be executed repeatedly are called the "range of the DO". The first time the statements in the range of DO are executed, "i" is initialized to the value " m_1 ", then at each succeeding time, "i" is increased by the value " m_3 ". At the end of the iteration, "i" is equal to the highest value that does not exceed " m_2 ", and then control passes to the statement following the statement numbered "n". The value " m_3 " is optional and when it is omitted, its value is assumed to be 1.

Thus, the DO statement causes the repeated execution of the statements within its range. Such a process could also be performed with other control statements, for example, with an IF statement.

```

J=0
3 READ(1,1)BASE,HEIGHT
1 FORMAT(F3.1,F2.1)
  AREA=BASE*HEIGHT/2.
  WRITE(3,2)AREA
2 FORMAT(1H0,'AREA= ',F8.2)
  J=J+1
  IF(J>10)3,5,5
5 CONTINUE
END

```

In this program, the integer variable "J" is used as a counter; that is, the statements between the READ statement and the IF statement are executed 10 times. With a DO statement, this program may be written as follows:

```

DO 5 J=1,10
READ(1,1)BASE,HEIGHT
1 FORMAT(F3.1,F2.1)
  AREA=BASE*HEIGHT/2.
  WRITE(3,2)AREA
2 FORMAT(1H0,'AREA= ',F8.2)
5 CONTINUE
END

```

Alternatively, the first statements may be written as

```

M=10
DO 5 J=1,M

```

or as

```
L=1
M=10
K=1
DO 5 J=L,M,K
```

In the above example, the DO statement causes the execution of the statements up to and including the statement numbered 5, thus resulting in the reading of 10 cards, and in the calculation and printing of 10 areas. During the last execution of the range (in this case, when J = 10), the DO is said to be satisfied, and there is a normal exit from the DO loop.

Rules for DO looping

(1) The indexing parameters of a DO statement (i , m_1 , m_2 , and m_3) must not be changed by any of the statements within the range of the DO loop. For example, the following is not allowed:

```
M=10
DO 5 I=1,M
M=2
.
.
.
5 CONTINUE
```

(2) The last statement in the range of a DO loop, i.e., statement numbered "n", cannot be a "GO TO", a "PAUSE", a "RETURN", a "STOP", and "IF", or another "DO" statement. The "CONTINUE" statement is most often used to circumvent this rule.

(3) There may be other DO statement(s) within the range of a DO loop and such a set is referred to as a nest of DO's. In nested DO's, the range of the innermost DO must be entirely within the range of the outer DO loop. For example,

```
DO 5 I=1,2
WRITE(3,7)
7 FORMAT(1H1)
DO 5 J=1,5
READ(1,1)BASE,HEIGHT
1 FORMAT(F3.1,F2.1)
AREA=BASE*HEIGHT/2.
WRITE(3,2)AREA
2 FORMAT(1HO,'AREA= ',F8.2)
5 CONTINUE
END
```

In this program, the outer (or first) DO starts the program off (I=1). The first WRITE statement causes the printer to move to the top of a new page, and then the second DO statement is satisfied, i.e., J = 1, J = 2, ..., J = 5. At this stage, five cards are read and five areas are calculated and printed. Control then passes back to the first DO statement, and the first WRITE statement causes the printer again to move to the top of a new page. The second DO statement is next to be satisfied again. At this step, the first DO statement is also satisfied (I = 2), and control is passed to the END statement.

(4) A transfer out of the range of a DO loop is permissible. For example,

```

DO 5 I=1,10
READ(1,1)BASE,HEIGHT
1 FORMAT(F3.1,F2.1)
  IF(BASE-99.9)3,4,4
3 AREA=BASE*HEIGHT/2.
5 WRITE(3,6)AREA
6 FORMAT(1H0,'AREA= ',F8.2)
4 CONTINUE
END

```

In this program, if, for example, the sixth card has 999 punched on it in columns 1 to 3, the IF statement would cause it to exit from the DO loop when I = 6. Note that statement number 6 is outside the DO loop, but this is allowed because the FORMAT statement is not executed and it may be placed anywhere in the source program.

(5) A transfer into the range of a DO loop from outside its range is not permitted. This rule also prohibits the transfer from the range of an outer DO into the range of an inner DO. However, it does permit the transfer from the range of an inner DO into the range of an outer DO, since such a transfer is within the range of the outer DO. An exception to these rules is that it is permitted to transfer out of a range of a DO to perform some subroutine, and then to transfer back to the same DO loop, providing that no change has been made in the values of the indexing parameters.

SPECIFICATION STATEMENTS

Specification statements provide the compiler with information about the nature of data in the source program, and supply the information required to allocate storage locations.

DIMENSION *statement*

Its general form is

DIMENSION $a_1(k_1), a_2(k_2), \dots, a_n(k_n)$

where: " a_1 ", " a_2 ", " a_n " are array names, and " k_1 ", " k_2 ", " k_n " are unsigned integer constants representing the maximum value of each subscript in the array.

For example, if the following one-dimensional array is punched on a card as 230 320 240 and if this array is named "W", then it can be read in with the following short program:

```
DIMENSION W(3)
READ(1,1)W(1),W(2),W(3)
1 FORMAT(3F5.1)
END
```

Note that in the DIMENSION statement the subscript quantity is 3, because the maximum number of elements in the array is 3. A further point of interest is that since the array is dimensioned, the order of elements need not be specified in the READ statement. For example,

```
DIMENSION W(3)
READ(1,1)W
1 FORMAT(3F5.1)
END
```

This program does exactly the same thing as the previous one. Therefore, when the order is not specified explicitly by either a READ or WRITE statement, the elements in a one-dimensional array are taken in a sequence, starting with the element corresponding to the subscript 1 and proceeding to the largest subscript as defined in the DIMENSION statement.

In basic FORTRAN, an array may be one-, two-, or three-dimensional. For example, the above one-dimensional array may be expanded into a two-dimensional one as follows:

```
230 320 240
235 326 242
```

The program for this array may be written as:

```
DIMENSION W(3,2)
READ(1,1)W(1,1),W(2,1),W(3,1),W(1,2),W(2,2),W(3,2)
1 FORMAT(3F5.1)
END
```

or alternatively

```
DIMENSION W(3,2)
READ(1,1)W
1 FORMAT(3F5.1)
END
```

These two programs give identical results. Similarly, when the order of elements is not specified explicitly for a two- (or three-) dimensional array, the elements are taken so that the first subscript is increasing first, then the second (then the third, if applicable).

The order of elements in an array may also be specified as follows: for one-dimensional array,

```
DIMENSION W(3)
READ(1,1)(W(I),I=1,3)
1 FORMAT(3F5.1)
END
```

For two-dimensional array,

```
DIMENSION W(3,2)
READ(1,1)((W(I,J),I=1,3),J=1,2)
1 FORMAT(3F5.1)
END
```

In the last example, the maximum values of "I" and "J" are 3 and 2, respectively, as specified in the DIMENSION statement. The subscript "I" is being incremented first from 1 to 3, and the corresponding values are read from the first data card (with 3F5.1 format). Next, "J" is incremented to 2, and "I" is again incremented from 1 to 3. This program may also be written with two "DO" statements:

```
DIMENSION W(3,2)
DO 5 J=1,2
DO 5 I=1,3
5 READ(1,1)W(I,J)
1 FORMAT(3F5.1)
END
```

In the general form of the DIMENSION statement, the subscript was defined as an unsigned integer constant, since it represents a set value. In the examples, however, it was shown that the value of a subscript may vary, provided it does not exceed its maximum value as specified in the DIMENSION statement. Thus, in general, a subscript is an integer quantity (or a set of integer quantities separated by commas) that is used to identify a particular element in an array. The value of a subscript must always be greater than 0. Subscripted quantities may be one of seven forms in basic FORTRAN:

- (1) an unsigned, nonsubscripted integer variable, e.g., L
- (2) an unsigned integer constant, e.g., 2
- (3) an unsigned, nonsubscripted integer variable + an unsigned integer constant, e.g., L+2
- (4) an unsigned, nonsubscripted integer variable - an unsigned integer constant, e.g., L-2
- (5) an unsigned integer constant x an unsigned nonsubscripted integer variable, e.g., 2*L
- (6) an unsigned integer constant x an unsigned nonsubscripted integer variable + an unsigned integer constant, e.g., 2*L+2
- (7) an unsigned integer constant x an unsigned nonsubscripted integer variable - an unsigned integer constant, e.g., 2*L-2

Finally, it should be noted that the DIMENSION statement describing data must precede any statement that refers to that data.

INTEGER specification statement

Its general form is

INTEGER a_1, a_2, \dots, a_n

where: " a_1 ", " a_2 ", " a_n " are variable names, or

INTEGER $a_1(k_1), a_2(k_2), \dots, a_n(k_n)$

where: " a_1 ", " a_2 ", and " a_n " are array or function names, and " k_1 ", " k_2 ", and " k_n " are unsigned integer constants representing the maximum value of each subscript in the array.

The INTEGER specification statement declares that a particular variable or array is in integer mode regardless of its initial character. For example, the variable named TREE and the array named W may be declared in integer mode by writing:

```
DIMENSION W(3,2)
INTEGER TREE,W
```

The array W may also be dimensioned in the specification statement:

```
INTEGER TREE,W(3,2)
```

Of course in this case, a "DIMENSION" statement for the "W" array is not needed.

REAL specification statement

Its general form is

REAL a_1, a_2, \dots, a_n or

REAL $a_1(k_1), a_2(k_2), \dots, a_n(k_n)$

Here, " a_1 ", " a_2 ", " a_n ", " k_1 ", " k_2 ", and " k_n " mean the same thing as in the case of the INTEGER specification statement. Thus, the REAL specification statement declares a particular variable or array to be in real mode. For example, the variable named NUMBER and the array named IW may be declared in real mode by writing

```
DIMENSION IW(3,2)
REAL NUMBER, IW
```

or simply

```
REAL NUMBER, IW(3,2)
```

Here again, the dimension information is given in the specification statement.

DOUBLE PRECISION specification statement

Its general form is

DOUBLE PRECISION a_1, a_2, \dots, a_n or

DOUBLE PRECISION $a_1(k_1), a_2(k_2), \dots, a_n(k_n)$

where: " a_1 ", " a_2 ", " a_n ", " k_1 ", " k_2 ", and " k_n " mean the same thing as in the case of the INTEGER and REAL specification statement. Hence, the DOUBLE PRECISION specification statement declares a particular variable or array to be in double-precision mode. For example:

```
DOUBLE PRECISION NUMBER, IW(3,2)
```

SOME FORTRAN LIBRARY SUBPROGRAMS

In scientific data processing, it is often necessary to obtain the square root or the logarithm of a number. To do this in a FORTRAN program, the relevant mathematical subprogram, which is generally stored on the disk during system generation, may simply be called upon by name. The following is a list of subroutines which are likely to be used.

(1) To compute the natural logarithm of a real number.

Entry name: ALOG

Form of use: ALOG(a) where "a" in parentheses is the argument, which may be a real number or a real arithmetic expression with a value greater than zero.

Examples: ALOG(2.)

ALOG(B)

ALOG(A+B/C)

(2) To compute the common logarithm of a real number.

Entry name: ALOG10

Form of use: ALOG10(a) where "a" is the same as in (1).

Examples: ALOG10(2.)

ALOG10(A)

ALOG10(A+B/C)

(3) To compute the natural logarithm of a double-precision number.

Entry name: DLOG

Form of use: DLOG(a) where "a" in parentheses is the argument, which may be a double-precision number or a double-precision arithmetic expression with a value greater than zero.

Examples: DLOG(2.)

DLOG(A)

DLOG(A+B/C)

(4) To compute the common logarithm of a double-precision number.

Entry name: DLOG10

Form of use: DLOG10(a) where "a" is the same as in (3).

Examples: DLOG10(2.)

DLOG10(A)

DLOG10(A+B/C)

(5) To compute the square root of a real number.

Entry name: SQRT

Form of use: SQRT(a) where "a" in parentheses is the argument, which may be a real number or a real arithmetic expression with a value equal to or greater than zero.

Examples: SQRT(2.)

SQRT(A)

SQRT(A+B/C)

(6) To compute the square root of a double-precision number.

Entry name: DSQRT

Form of use: DSQRT(a) where "a" in parentheses is the argument, which may be a double-precision number or a double-precision arithmetic expression with a value equal to or greater than zero.

Examples: DSQRT(2.)

DSQRT(A)

DSQRT(A+B/C)

There are a number of other mathematical subprograms available, such as the computation of the sine or cosine of an argument representing a number. To obtain more information on these, the reader should consult the IBM manual.

PROGRAMMING CONSIDERATIONS

In this text, subprograms are not dealt with, since the primary objective of this paper is to introduce the reader to programming. It is assumed that those wishing to reach a more sophisticated or advanced level in programming are now sufficiently equipped to use the IBM-supplied manuals. For the beginner, it is best to practice with single programs more extensively before attempting to write subprograms.

In the remainder of this paper, the Disk Operating System will also be introduced and the control cards necessary to run a program through the computer will be given.

DISK OPERATING SYSTEM

In this section, an attempt will not be made to explain in detail the Disk Operating System (DOS), but rather to show how certain types of jobs may be carried out with disks. In particular, information given here refers to the IBM 360/25 system with two IBM 2311 disk storage units, as used by the Ontario Region. Furthermore, only the storage of data on the disk and the retrieval of data from the disk is being presented here. Those readers wishing information about how to store programs or subprograms in the relocatable library or core image library, or those wishing to understand the DOS, should consult the IBM manuals.

In cases where large volumes of data are to be processed, and especially if these data are to be put through internal storage several times, the available core storage in the computer may not be large enough to handle the job. Even if it is large enough, the repeated loading of, for example, 10,000 cards can become an unattractive or an uneconomical proposition. In such cases, it is best to store the data on the disk.

To be able to do this, certain information must be supplied to the computer regarding the size of the data and their positions on the disk. This requires some knowledge of the structure of the disk.

As stated above, the computer used by Ontario Region has an IBM 2311 disk storage unit attached to it. A disk pack, which consists of six disks, is mounted on each storage unit or disk drive. These disks, looking like phonograph records, are 14 inches in diameter and are spaced about 1 inch apart. Each disk has two faces, upper and lower. Thus, in a disk pack, there are 12 faces. However, the upper face of the top disk and the lower face of the bottom disk, i.e., the outside faces are not used because they are too vulnerable to damage. This leaves 10 available faces which are numbered 0 to 9 starting with the lower face of the top disk (the upper face of the bottom disk is, therefore, numbered 9). Each disk face contains 200 circular areas or tracks on which information can be recorded. These 200 tracks over 10 disk faces form 200 cylinders, thus each cylinder contains 10 tracks. Cylinders are numbered, starting with 0 at the outer edge and ending with 199 at the inside of the disk. On each track, 3,625 bytes of information may be stored. Thus the capacity of the entire pack is about 7.5 million bytes.

When the disk pack is spinning at a relatively high speed on the disk drive, five access arms, each containing two READ-WRITE heads, move between the disks in such a way that the 10 heads are positioned above each other in a cylinder. Therefore, for example, when the upper head is positioned at track 0 on disk surface 0, the rest of the heads are also positioned at cylinder 0 over their respective disk surfaces.

Information may be stored on or retrieved from the disk in a sequential manner (Sequential Access), or by going directly to the relevant cylinder (Direct Access). Storing data on a disk pack can be an involved and a complex operation, especially from a beginner's point of view. Therefore in this paper, only one method is given which is the Direct Access. This might not always be the most efficient one, but it is the one most often used in the Ontario Region for the programming of research data.

Before data can be manipulated, during Direct Access, between internal storage and the disk pack, it must be defined in some way. In a FORTRAN source program, this is done by the "DEFINE FILE" statement.

DEFINE FILE statement

Its general form is

DEFINE FILE $i_1(m_1, r_1, f_1, v_1), i_2(m_2, r_2, f_2, v_2), \dots, i_n(m_n, r_n, f_n, v_n)$

where: " i_1 ", " i_2 ", " i_n " are integer constants, representing data set reference numbers. The user should obtain the relevant number

from the personnel of the computer centre. For the computer used by the Ontario Region, any of the following integer constants may be used: 7, 8, 9, 10, 11, and 12 (if the user requires more than six numbers, he should consult with the Regional Biometrician).

" m_1 ", " m_2 ", " m_n " are integer constants, specifying the maximum number of records in the data set.

" r_1 ", " r_2 ", " r_n " are integer constants, specifying the maximum size of each record in the data set. This may be measured in bytes or in words (number of bytes divided by four and rounded to the next highest integer). At this stage, one must remember that an integer variable or constant and a real variable or constant occupies four storage locations or four bytes, whereas a double-precision variable or constant takes eight bytes. For example, if the values of two integer variables and of three real variables are punched on a card, then this can be taken as a record whose length is calculated as follows:

record length = number of variables \times 4 = 20

Similarly, for two integer and three double-precision variables, the record length is $2 \times 4 + 3 \times 8 = 32$.

" f_1 ", " f_2 ", " f_n " specifies that the data set is to be read or written either with or without format control. The following three letters are used here:

E indicating that I/O will be formatted,
 U indicating that I/O will be unformatted, and
 L indicating that I/O will be mixed--some formatted, and some unformatted. This is perhaps the most convenient one to use.

" v_1 ", " v_2 ", " v_n " represents a nonsubscripted integer variable, called the associated variable, referring to the "index" number of each record. Alternatively, consider that the records in the data set are numbered from 1 up to and including " m_1 ", " m_2 ", " m_n ", i.e., the maximum record number.

For example, if 10 variables (some integer, some real) are punched per card, and there are 100 such cards, the DEFINE FILE statement may be written as

DEFINE FILE 7(100,40,L,INDEX)

The DEFINE FILE statement in the source program must precede any input/output statement referring to the data it defines.

Now that the data for the computer is defined, it may be manipulated between internal storage and the disk pack. This is done by READ and WRITE statements.

Direct Access WRITE statement

The D.A. WRITE statement causes data to be transferred from internal storage to a direct access device (in this case, to the disk). Therefore, data must be put into internal storage first. For example, if the input is through cards, then the data can be transferred into internal storage (or read in) with an ordinary READ statement. Such data may then be put on the disk with a D.A. WRITE statement. Its general form is:

WRITE(i'k,j)list or simply
WRITE(i'k)list

where: "i" is an unsigned integer constant or integer variable that represents the data set reference number as given in the DEFINE FILE statement (corresponding to "i₁","i₂","i_n").

"k" is an integer expression that represents the relative position of a record within the data set associated with "i" (refers to "v₁","v₂","v_n" in the DEFINE FILE statement).

"j" is optional and, if given, is the statement number of the FORMAT statement that describes the data being written. It is often simpler to omit j.

"list" refers to the possible list of variable names, separated by commas.

For example, if there are six variables (A,B,C,D,E, and L) per card, and there are 100 such cards, the following program would store these data on the disk.

```
DEFINE FILE 7(100,24,L,INDEX)
INDEX=1
DO 5 I=1,100
READ(1,2)A,B,C,D,E,L
2 FORMAT(5F10.0,I10)
5 WRITE(7'INDEX)A,B,C,D,E,L
END
```

Direct Access READ statement

The D.A. READ statement causes data to be transferred from a direct access device (a disk) into internal storage. Its general form is:

```
READ(i'k,j)list  or simply
READ(i'k)list
```

where: "i" is an unsigned integer constant or integer variable that represents the data set reference number as given in the DEFINE FILE statement.

"k" is an integer expression that represents the relative position of a record within the data set associated with "i".

"j" is optional and, if given, is the statement number of the FORMAT statement that describes the data being read.

"list" refers to the possible list of variable names, separated by commas.

For example, the following program would cause the printing of the data stored on the disk by the above program.

```
DEFINE FILE 7(100,24,L,INDEX)
INDEX=1
DO 10 I=1,100
  READ(7'INDEX)A,B,C,D,E,L
10 WRITE(3,8)A,B,C,D,E,L
  8 FORMAT(1H0,5F12.0,I11)
  END
```

When programming for direct access files, the area(s) of file(s) on the disk must be preformatted. This again can be a rather complicated procedure. Therefore, only a simple and practical method, which is applicable to the Ontario Region's facilities, is presented here. For preformatting, the following program may be used.

```
// JOB CLEARDSK
// DLBL UOUT,'FILE NAME'
// EXTENT SYS004,111111,1,,1800,190
// EXEC CLRDSK
// UCL B=(K=0,D=24),X'00',0Y,E=(2311)
// END
```

However, the above program needs to be modified in order to accommodate the particular set of data it is referring to. Thus, the following changes may need to be made:

- (1) On the third card or EXTENT card, SYS004 (which is a logical unit name) refers only to data set reference number 7. If more data set reference numbers are used, the corresponding logical unit names are:

Data set reference number Logical unit name

7	SYS004
8	SYS005
9	SYS006
10	SYS007
11	SYS008
12	SYS009

For example, if we use data set reference number 9, the corresponding logical unit name in the EXTENT card is SYS006.

The last two integer numbers on the EXTENT card refer to relative track number and to number of tracks, respectively. Users in the Ontario Region may leave these two numbers unchanged when creating only temporary files. When creating protected files (that is, wishing to protect the storage of data on the disk for a certain period of time) users should consult with the Regional Biometrician.

(2) On the fifth card of UCL card, "D" refers to the length of a record. Therefore, it is equal to the value given for "r₁", "r₂", "r_n" in the DEFINE FILE statement. For example, if there are six variables (integer, real, or both) per record, D=24.

After the disk is formatted for a particular set of data, numerous jobs may be run on that set of data without formatting it again with the above CLEARDISK UTILITY program. However, for each such job, two cards need to be inserted between the // EXEC LNKEDT and the // EXEC cards to provide information about the location of the particular data on the disk. These cards are:

```
// DLBL IJSYS04,'FILE NAME'
// EXTENT SYS004,111111,1,,1800,190
```

On the first card or DLBL card, IJSYS04 is the DOS file name, corresponding to FORTRAN data set reference number 7. Again, the cross reference between the two are as follows:

Data set reference number DOS file name

7	IJSYS04
8	IJSYS05
9	IJSYS06
10	IJSYS07
11	IJSYS08
12	IJSYS09

The second card or EXTENT card is the same as the one used in the CLEARDISK UTILITY program.

As mentioned earlier, FORTRAN source programs alone are not accepted by the computer. For the execution of a job, monitor or control cards are needed as well. Some of these are listed in Appendix C.

BIBLIOGRAPHY

GERMAIN, C.B. 1967. Programming the IBM 360. Prentice-Hall Inc. New York. 366 p.

IBM 1969. IBM System/360. Basic FORTRAN IV Language. International Business Machines Corporation, Systems Reference Library, File No. S360-25, GC28-6629-2. 90 p.

IBM 1969. IBM System/360. Disk and Tape Operating Systems. Basic FORTRAN IV. Programmer's Guide. International Business Machines Corporation, Systems Reference Library, File No. S360-25, Form C24-5038-3. 91 p.

IBM 1966. IBM System/360. Disk Operating System. System Control and System Service Programs. International Business Machines Corporation, Systems Reference Library, File No. S360-36, GC24-5036-4. 178 p.

McCRACKEN, D.D. 1965. A guide to FORTRAN IV programming. John Wiley & Sons, Inc. New York. 151 p.

APPENDICES

APPENDIX A

(For Answers, see Appendix D)

Exercise 1. Integer constants:

(A) Write the following numbers as FORTRAN integer constants.

(1) 1,222 (2) 258.0 (3) +1. (4) -3. (5) 737,438.0

(B) Which of the following numbers are valid integer constants?

(1) 3,728 (2) \$132.75 (3) 25 (4) 9 (5) 1234567

(C) Why are the following numbers not valid integer constants?

(1) 4.37 (2) 3,426 (3) -.25 (4) 3A7 (5) .000895

Exercise 2. Real constants:

(A) Write the following numbers as FORTRAN real constants.

(1) 375 (2) -3.0 (3) 4,382 (4) \$375.82 (5) 32

(B) Which of the following numbers are valid real constants?

(1) 5597 (2) 375.95 (3) 192 (4) 4,378 (5) 37.

(C) Why are the following numbers not valid real constants?

(1) 289 (2) E3-79 (3) 9278E (4) 000 (5) \$.78

Exercise 3. Integer variables:

(A) Write the following names as FORTRAN integer variables.

(1) jack (2) tree (3) dutch (4) elms (5) sick

(B) Which of the following names are valid integer variables?

(1) I (2) LIKE (3) ICE-CREAM (4) \$COST (5) MONEY

(C) Why are the following names not valid integer variables?

(1) ID. (2) OWES (3) DAVID (4) THREE (5) DOLLARS

Exercise 4. Real variables:

(A) Write the following names as FORTRAN real variables.

(1) input (2) xyz (3) Help (4) Pine (5) ELM

(B) Which of the following names are valid real variables?

(1) JIM (2) AND (3) MARY (4) LIKE (5) TREE

(C) Why are the following names not valid real variables?

(1) ILIKE (2) JUICE (3) APPLE-PIE (4) 27I (5) MAPLE

Exercise 5. Arithmetic expressions:

(A) Write a FORTRAN arithmetic expression to compute

$$(1) \frac{a}{b} \quad (2) \frac{a}{-b} \quad (3) \frac{6a}{b} \quad (4) \frac{ab}{c} \quad (5) \frac{4}{3} r^3 \quad (6) (a+b)^2$$

$$(7) (a+b)^3 \quad (8) \frac{a+b}{c+d} \quad (9) \frac{a+b}{c+d}^2 + x^2 \quad (10) x^2 + xy + y^2$$

Exercise 6. Arithmetic statements:

(A) Write a FORTRAN arithmetic statement to compute

$$(1) \frac{b}{b} = c \quad (2) c = \frac{a}{-b} \quad (3) \frac{ab}{d} = c \quad (4) a+b = c+d \quad (5) x^2 = y$$

(B) Given the values (A=2.), (B=3.), (C=4.), (I=5), and (J=1), evaluate the following FORTRAN expressions

$$(1) D=A+B \quad (2) E=A*B \quad (3) F=A*B/C \quad (4) K=I**2 \quad (5) G=J/I$$

$$(6) L=I*A/(I+1) \quad (7) Q=A**B/C \quad (8) OH=B*C/I \quad (9) MOH=B*C/I$$

$$(10) Y=I**J$$

Exercise 7. Input/output statements:

(A) Write the necessary statements which would cause the reading of three variables (A, B, and I) from a card. The values for A, B, and I are punched in columns 1-5 (with one decimal), 6-10 (with 2 decimal places), and in columns 11-15, respectively.

(B) Write the necessary statements which would

- (1) cause the printer to be positioned on the first line of a new page
- (2) cause the printing of THIS IS A HEADING on the second line of a new page
- (3) cause the printing of A FORMAT EXERCISE WITH TRICKS on the same line as the printer is positioned, but starting in column 25.

Exercise 8. Control statements:

(A) Given 10 cards with values for BASE and HEIGHT punched on each in columns 1-5 (with 1 decimal), and in columns 6-10 (with 2 decimal places), respectively,

- (1) Write a program to calculate the values of AREA, where $AREA=BASE*HEIGHT/2$. and test for the last card (on the eleventh card, 9999. is punched in columns 11-15). Print the results on a new page under the heading:

AREAS OF TRIANGLES FOR EXERCISE 8.

- (2) Write a program to calculate the values of AREA, where $AREA=BASE*HEIGHT/2$. and test for the last card (the eleventh card is blank). Print the first five results on a new page with the heading AREAS OF THE FIRST 5 TRIANGLES FOR EXERCISE 8 and use double spacing. Then, print the second five results on a separate page with single spacing and under the heading AREAS OF TRIANGLES (6-10) FOR EXERCISE 8.

APPENDIX B

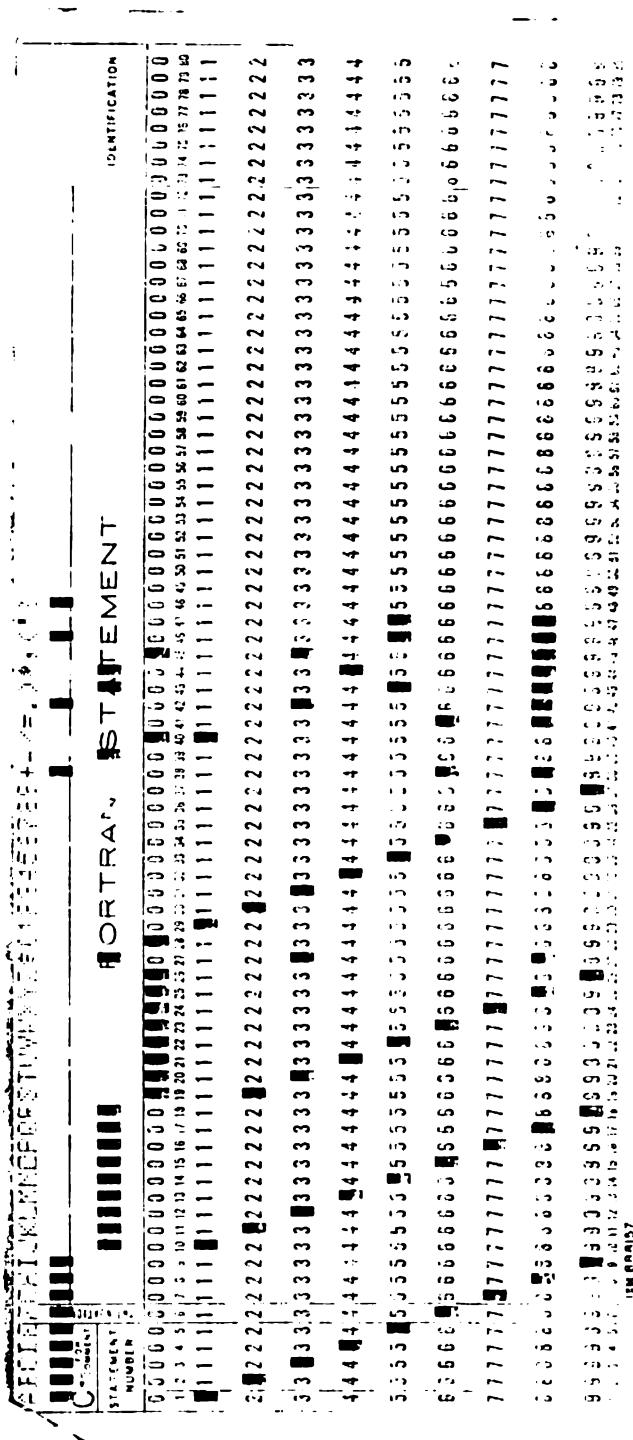


Figure 1. A punched card with alphabetic, numeric, and special characters.

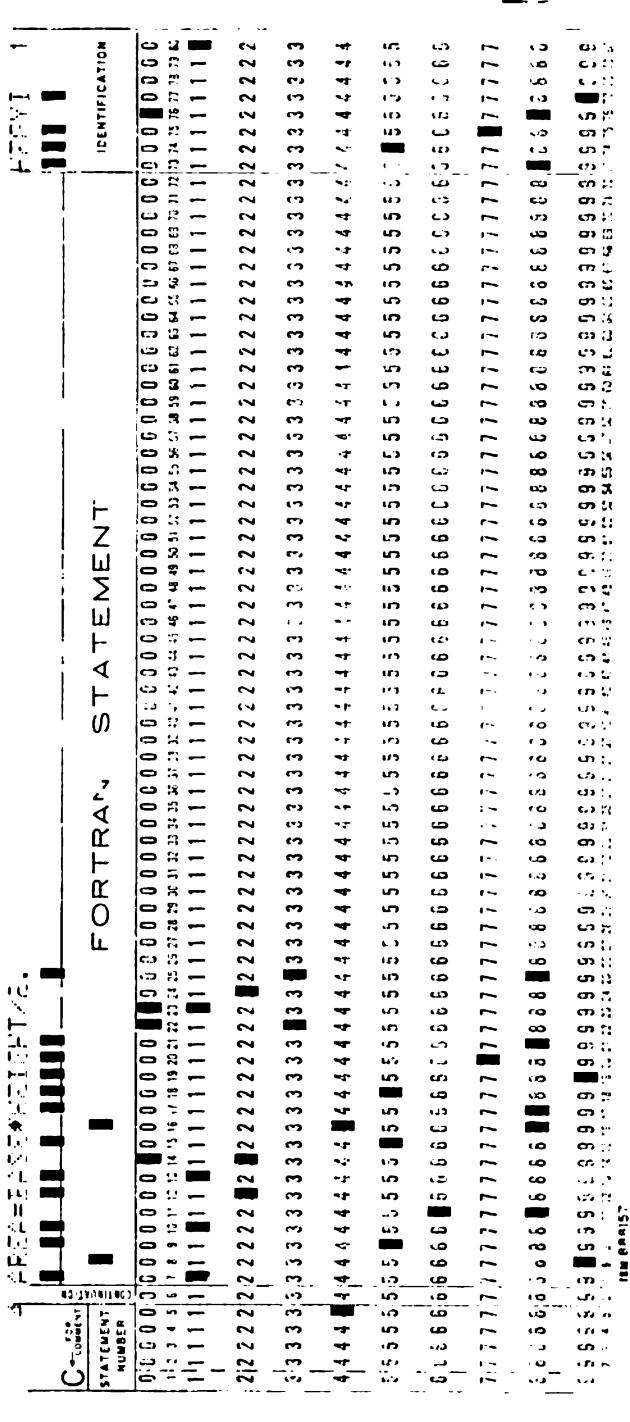


Figure 2. A punched card with a FORTRAN statement.

FORTRAN CODING FORM

```

1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25 26 27 28 29 30 31 32 33 34 35 36 37 38 39 40 41 42 43 44 45 46 47 48 49 50 51 52 53 54 55 56 57 58 59 60 61 62 63 64 65 66 67 68 69 70 71 72 73 74 75 76 77 78 79 80
IV. JOB. TRIANGLE
M 2 // OPTIMIZATION LINK
J // EXEC FORTRAN
AC AREA OF TRIANGLE PROGRAM.
S
5. READ(L,1) BASE, HEIGHT
6. FORMAT(F3.1,F2.1)
7. AREA=BASE*HEIGHT/2
8. WRITE(L,2) AREA
9. FORMAT(IH1,2E15.15)
10. END
11. END
12. END
13. EXEC LINKED
14. EXEC
15. EXEC
16. EXEC
17. EXEC
18. EXEC
19. EXEC
20. EXEC

```

Figure 3. Coding sheet for the AREA OF TRIANGLE program. Note that each line represents a card. Cards marked M are the monitor or control cards needed to put the program through the computer; cards marked S make up the source program; and the one marked D is a data card. This is a typical set-up of cards for a job to be run through an IBM 360/25 computer.

APPENDIX C

Examples of job control cards for use on an IBM 360/25

1. Compilation of a FORTRAN source program

```
// JOB NAME
// EXEC FORTRAN
.
.
.
source program
.
.
.
/*
/&
```

Note that the first "/" starts in column 1. On the first control card (// JOB NAME), "NAME" is the title or name of the job and it may be from one through eight characters.

2. Compilation and execution of a FORTRAN source program

```
// JOB NAME
// OPTION LINK
// EXEC FORTRAN
.
.
.
source program
.
.
.
/*
// EXEC LNKEDT
// EXEC
.
.
.
data
.
.
.
/*
/&
```

3. Compilation of a FORTRAN source program and production of an object deck (machine language translation of a source program)

```
// JOB NAME
// OPTION DECK
// EXEC FORTRAN
.
.
.
source program
.
.
.
/*
/&
```

4. Execution of an object module

```
// JOB NAME
// OPTION LINK
INCLUDE
.
.
.
object deck (obtained from previous job set up (3))
.
.
.
/*
// EXEC LNKEDT
// EXEC
.
.
.
data
.
.
.
/*
/&
```

Note that if several sets of data are to be analysed with the same program, it is more efficient to convert the source program into an object module and then to use the above set up.

APPENDIX D

(Answers to Exercises in APPENDIX A)

Exercise 1.

(A) (1) 1222 (2) 258 (3) 1 (4) -3 (5) 737438
 (B) (3) 25 (4) 9 (5) 1234567
 (C) (1) decimal point (2) comma (3) decimal point (4) alphabetic character (5) decimal point

Exercise 2.

(A) (1) 375. (2) -3.0 (3) 4382. (4) 375.82 (5) 32.
 (B) (2) 375.95 (5) 37.
 (C) (1) no decimal point (2) E should follow 3, even then, it would be too large (3) integer constant after E is missing (4) no decimal point (5) alphabetic character in numeric field

Exercise 3.

(A) (1) JACK (2) ITREE (3) IDUTCH (4) IELMS (5) ISICK
 (B) (1) I (2) LIKE (5) MONEY
 (C) (1) special characters in name (2) real variable name (3) real variable name (4) real variable name (5) more than six characters

Exercise 4.

(A) (1) AINPUT (2) XYZ (3) HELP (4) PINE (5) ELM
 (B) (2) AND (5) TREE
 (C) (1) integer variable name (2) integer variable name (3) too many characters and a special character in the name (4) the first character is numeric (5) integer variable name

Exercise 5.

(A) (1) A/B (2) A/(-B) (3) 6*A/B (4) A*B/C (5) (4**3/3)*R

(6) $(A+B)^{**2}$ (7) $(A+B)^{**3}$ (8) $(A+B) / (C+D)$
 (9) $((A+B^{**2}) / (C+D)) + X^{**2}$ (10) $X^{**2} + X^{**Y} + Y^{**2}$

Exercise 6.

(A) (1) $C=B/B$ (2) $C=A/(-B)$ (3) $C=A*B/D$ (4) $A=C+D-B$ or $B=C+D-A$
 (5) $Y=X^{**2}$

(B) (1) $D=5.$ (2) $E=6.$ (3) $F=1.5$ (4) $K=25$ (5) $G=0.$ (6) $L=1$
 (7) $O=2$ (8) $OH=2$ (9) $MOH=2$ (10) $Y=5$

Exercise 7

```
(A)  READ(1,1)A,B,I
      1 FORMAT(F5.1,F5.2,I5)

(B)  (1)    WRITE(3,2)          (2)    WRITE(3,3)
      2 FORMAT(1H1)            3 FORMAT(1H1,/, ' THIS IS A HEADING')

      (3)    WRITE(3,4)
      4 FORMAT(1H+,24X,29HA FORMAT EXERCISE WITH TRICKS)
```

Exercise 8.

(A) (1) WRITE(3,1)
 1 FORMAT(1H1,'AREAS OF TRIANGLES FOR EXERCISE 8.')
 6 READ(1,2)BASE,HEIGHT,TEST
 2 FORMAT(F5.1,F5.2,F5.0)
 IF(TEST-9999.)3,4,3
 3 AREA=BASE*HEIGHT/2.
 WRITE(3,5)AREA
 5 FORMAT(1H ,12X,F8.2)
 GO TO 6
 4 CONTINUE
 END

(2) N=0
 WRITE(3,1)
 1 FORMAT(1H1,'AREAS OF THE FIRST 5 TRIANGLES FOR EXERCISE 8.')
 11 READ(1,2)BASE,HEIGHT
 2 FORMAT(F5.1,F5.2)
 IF(BASE)3,10,3
 3 N=N+1
 AREA=BASE*HEIGHT/2.
 IF(N-6)5,8,7
 5 WRITE(3,6)AREA

```
6 FORMAT(1H 0,19X,F8.2)
  GO TO 11
8 WRITE(3,9)
9 FORMAT(1H1,'AREAS OF TRIANGLES (6-10) FOR EXERCISE 8.')
7 WRITE(3,4)AREA
4 FORMAT(1H ,12X,F8.2)
  GO TO 11
10 CONTINUE
END
```